Geoprocessing with ModelBuilder and Python

Jeremiah Lindemann
ESRI Denver

AGIC 2005 Conference
Prescott, AZ

Workshop agenda

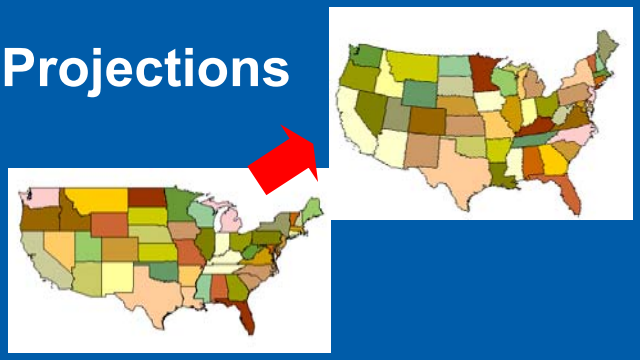
- Geoprocessing
- Introduction to Modelbuilder
  - Building and running models
  - Exporting models
- Introduction to Python
  - Scripting overview
  - Python language
  - Batch Processing
  - Making Scripts Dynamic

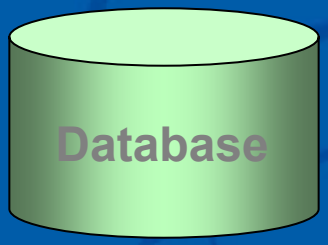
Geoprocessing and Models

What is geoprocessing?

- Perform a variety of geographic based tasks

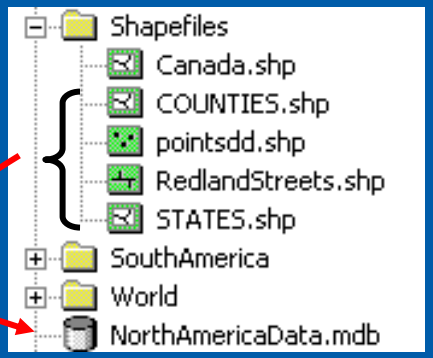
Projections



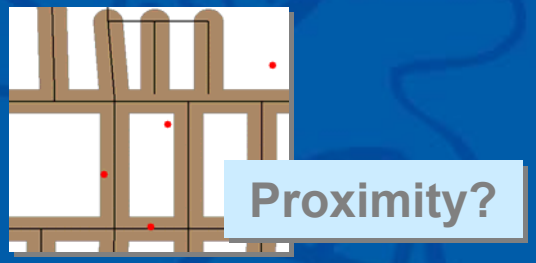


CAD **GDB**

Conversion



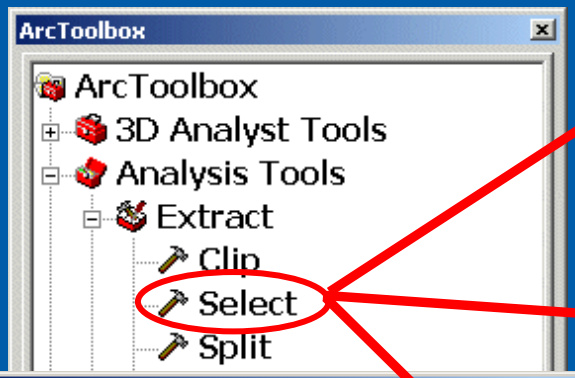
Data Management

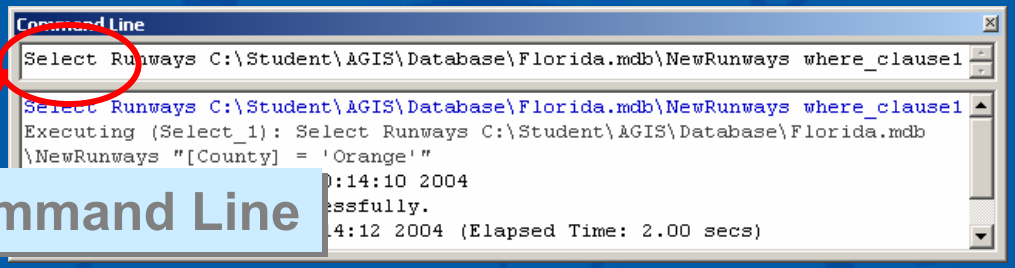


Spatial Analysis

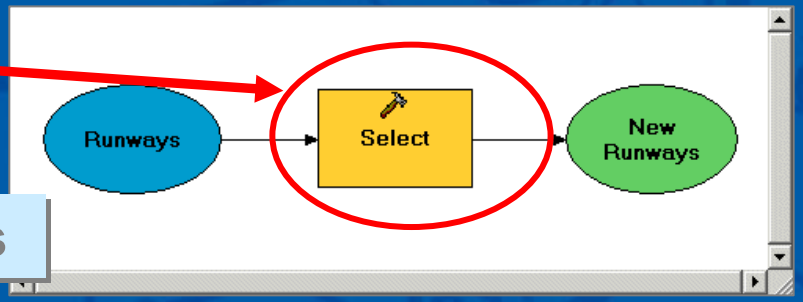
Geoprocessing framework

- Multiple environments

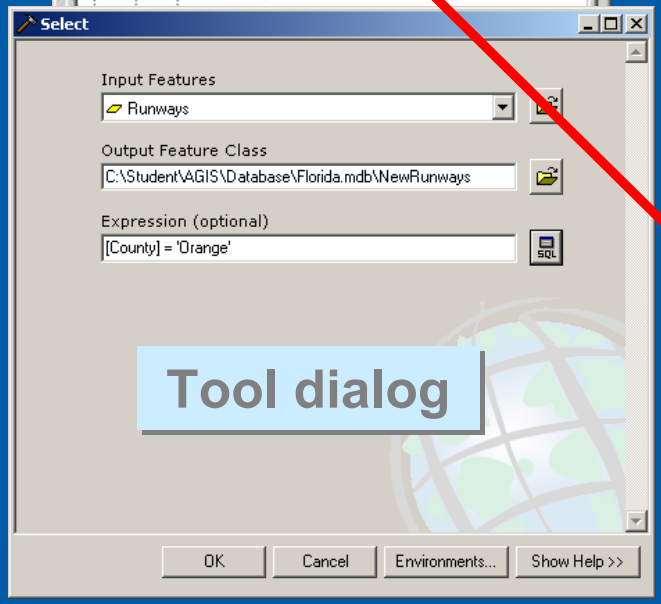




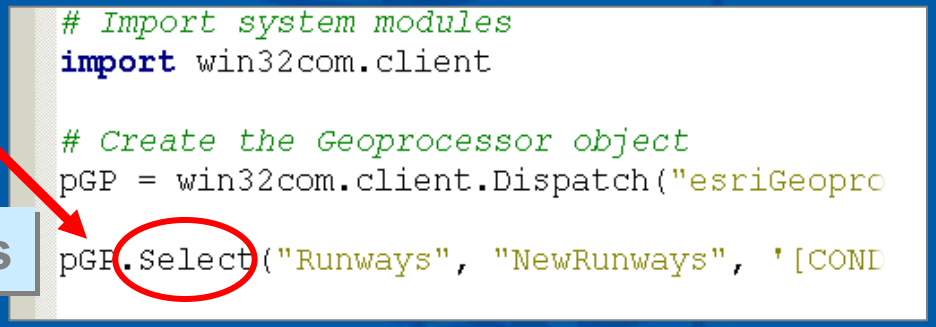
Command Line



Models



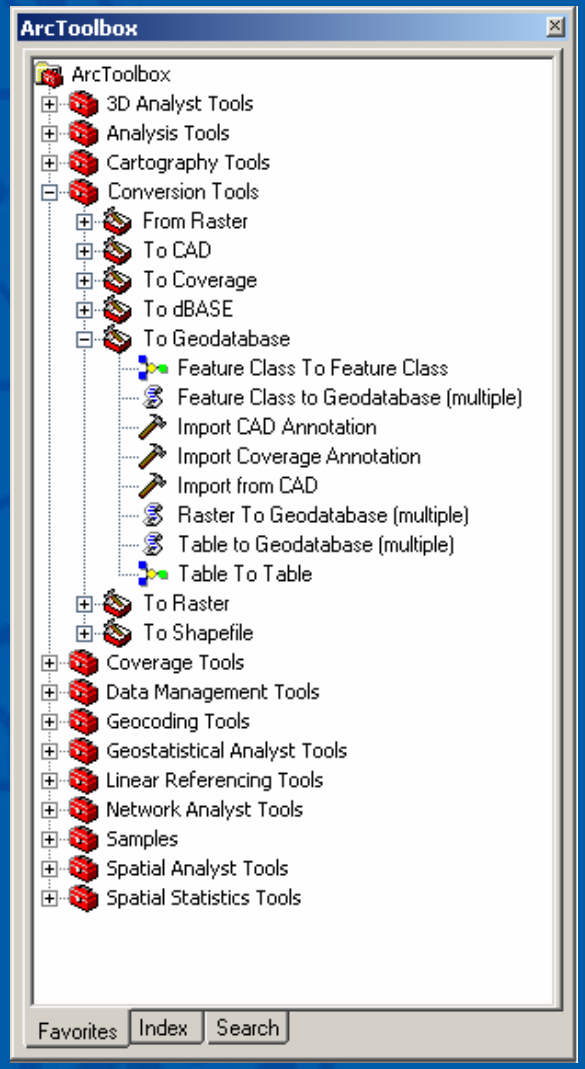
Tool dialog



Scripts

ArcToolbox

- Dockable window
- Toolboxes and tools: functionally ordered tree view
- Number of tools depends on license
- Tools can be used in models



ArcGIS extensions

Business Analyst
Spatial Analyst
3D Analyst
Geostatistical Analyst
Network Analyst
Data Interoperability

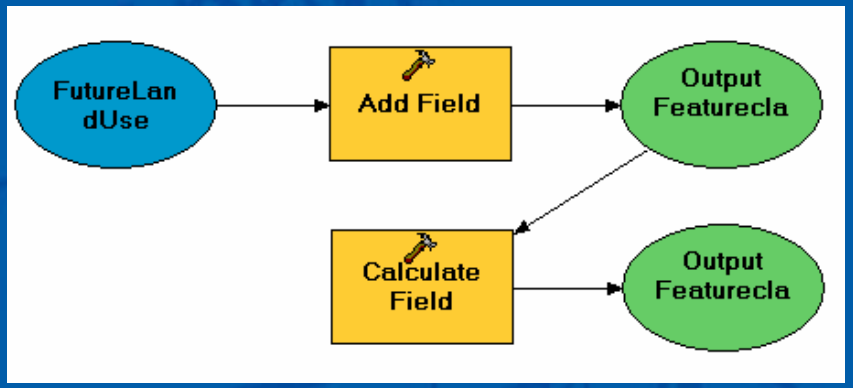
Tools and licensing

- The tools vary depending on license/extensions
  - ArcView: 159 tools
  - ArcEditor: 184 tools
  - ArcInfo: 216 tools

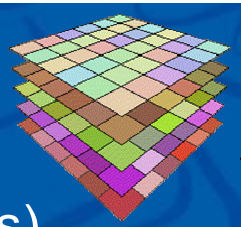
  - More tools available with additional extensions

Types of models

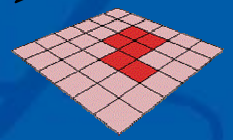
- Repetitive tasks
  - Minimize *grunt work*
  - Efficiently execute frequently used tools



- Suitability models
  - Use to find best location (businesses, vineyards, evacuation centers)

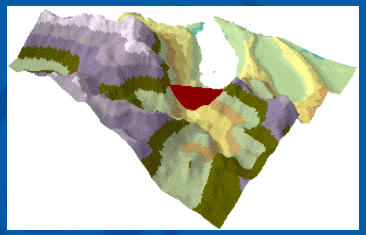


GIS data layers



Best store location

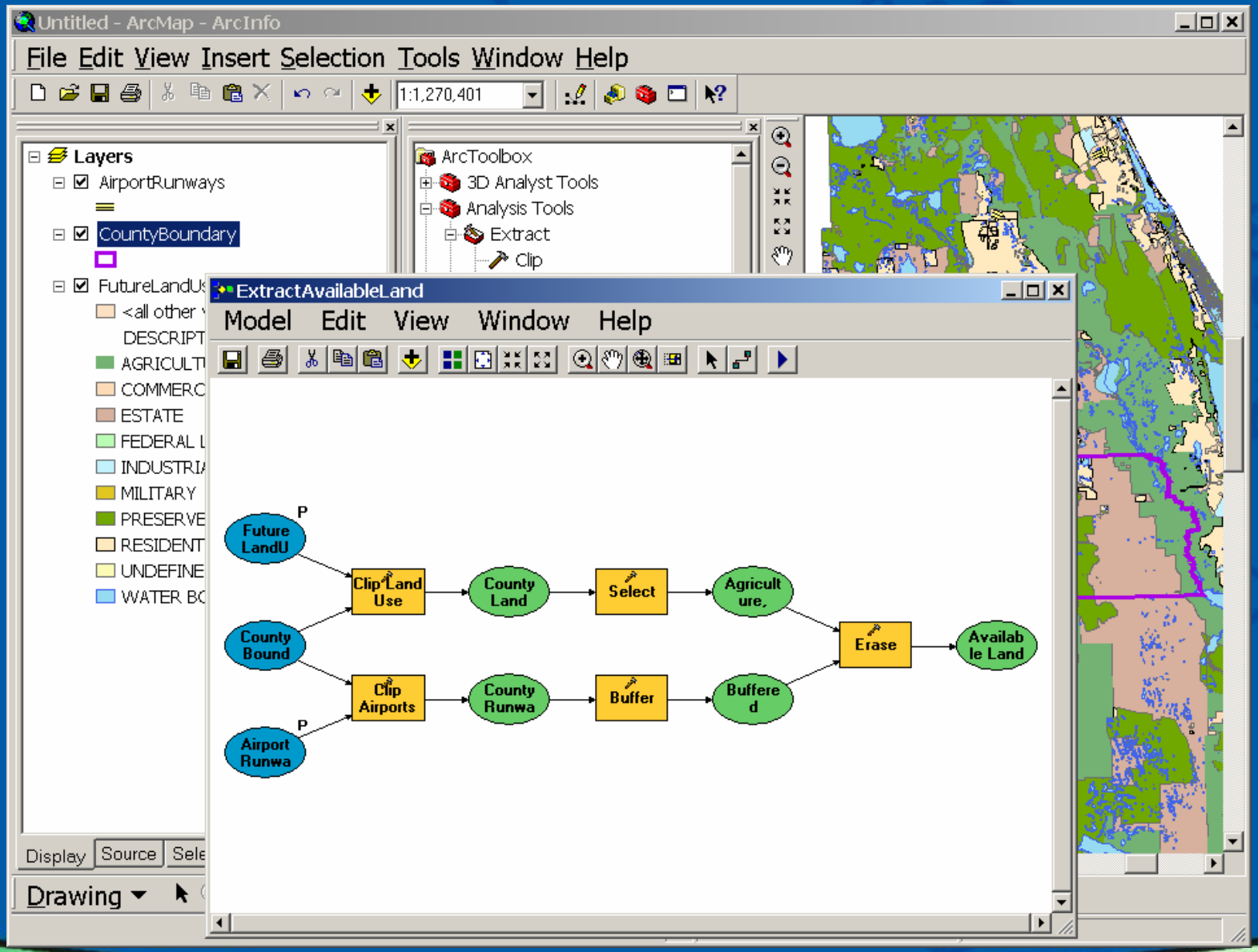
- Process models
  - Show the landscape as conditions change (fire spreads, rivers flood, oil slicks move)



Filling a reservoir

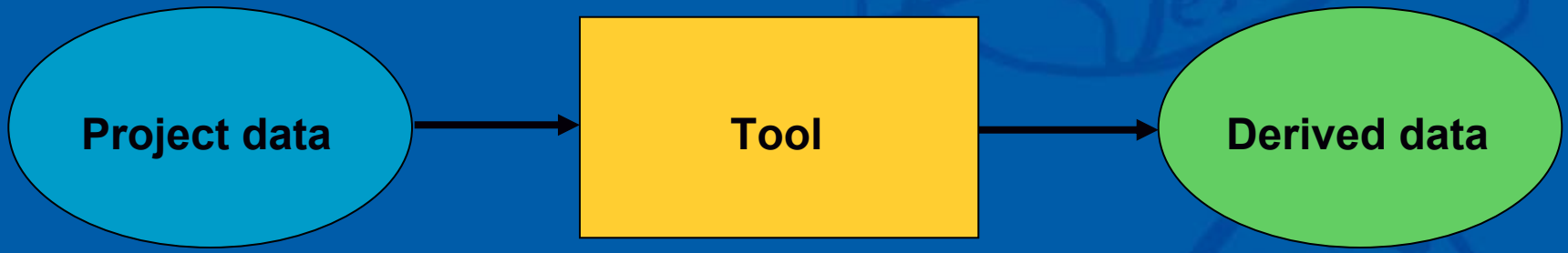
Introduction to Modelbuilder

ModelBuilder



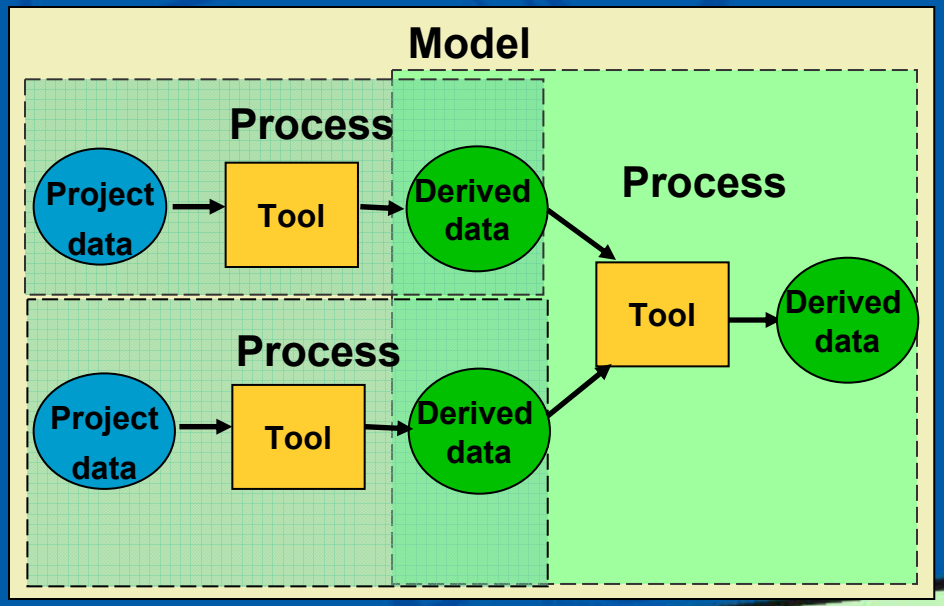
Why use ModelBuilder?

- Fast analysis
- Re-execute the same model, slightly changing parameters to see how end results differ
- Complex analysis
- Graphical documentation of work



Model elements

- Project data: Data that exists before model is run
  - Blue oval
- Tool: Operation performed on input data
  - Yellow-orange rectangle
- Derived Data: Output data created by a function
  - Green oval
- Process: Set of elements
  - Run one process at a time or all at once

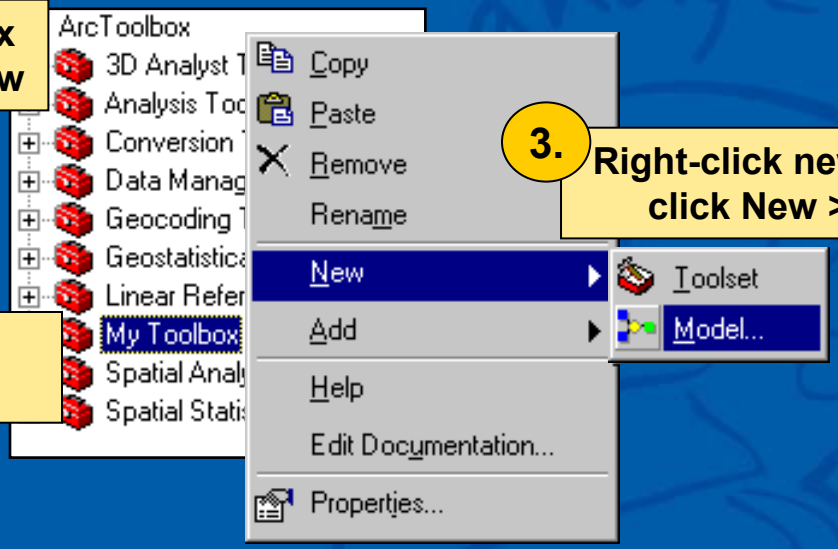


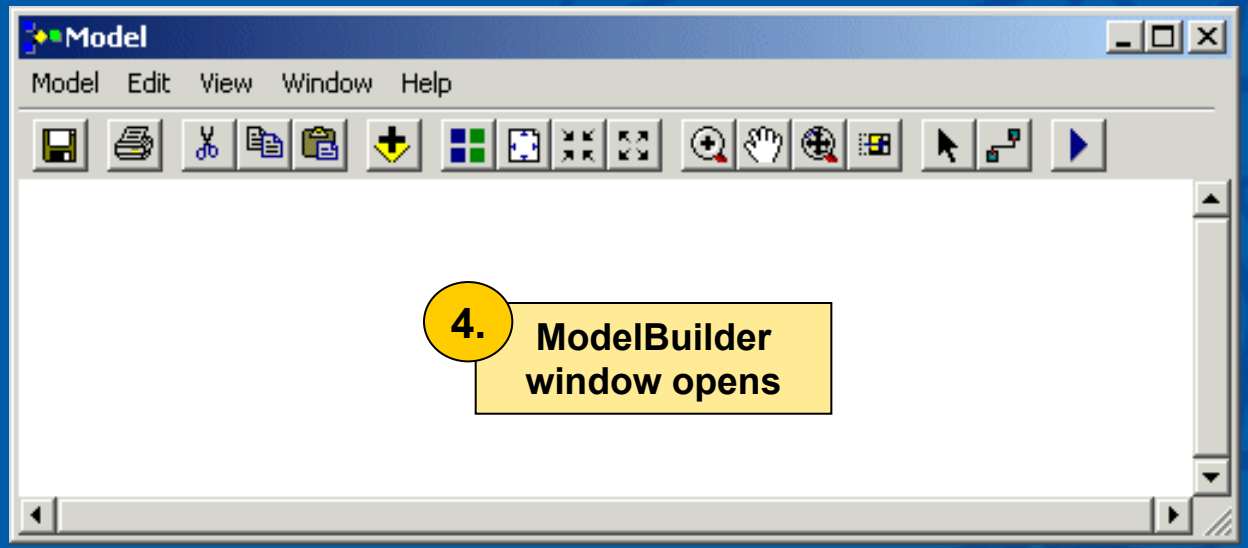
Creating a model

1. Add ArcToolbox dockable window

2. Create a new user toolbox

3. Right-click new toolbox; click New > Model

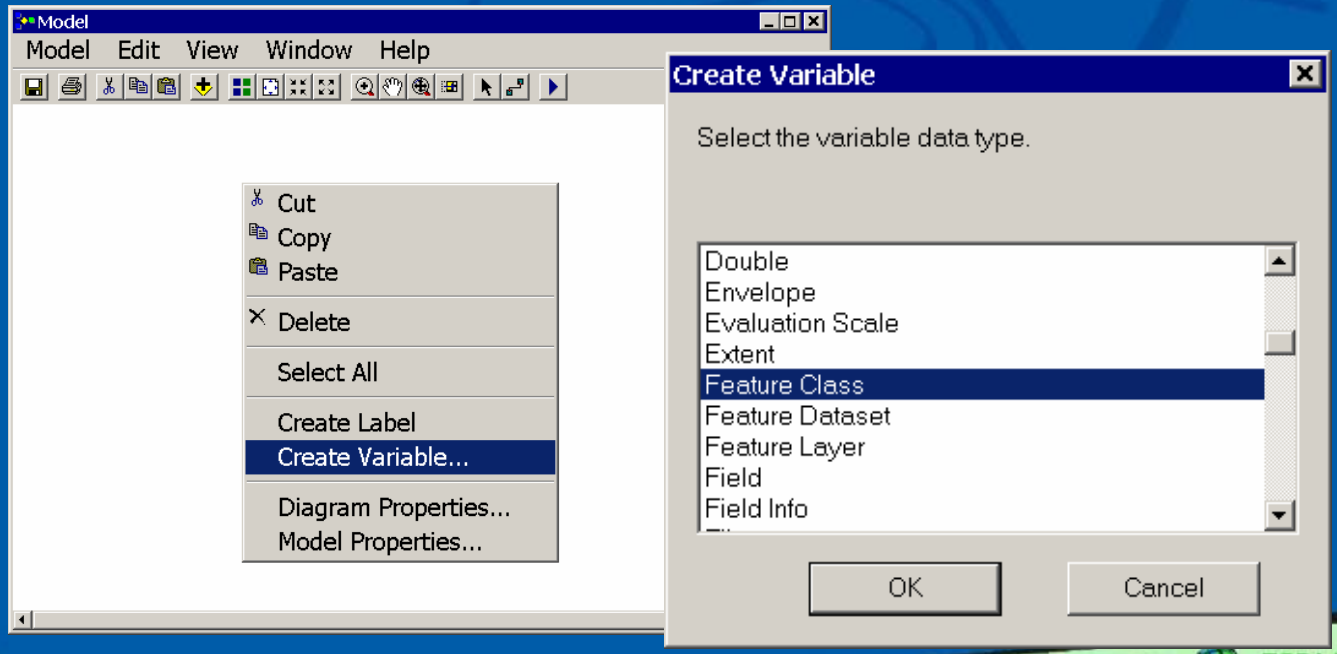




4. ModelBuilder window opens

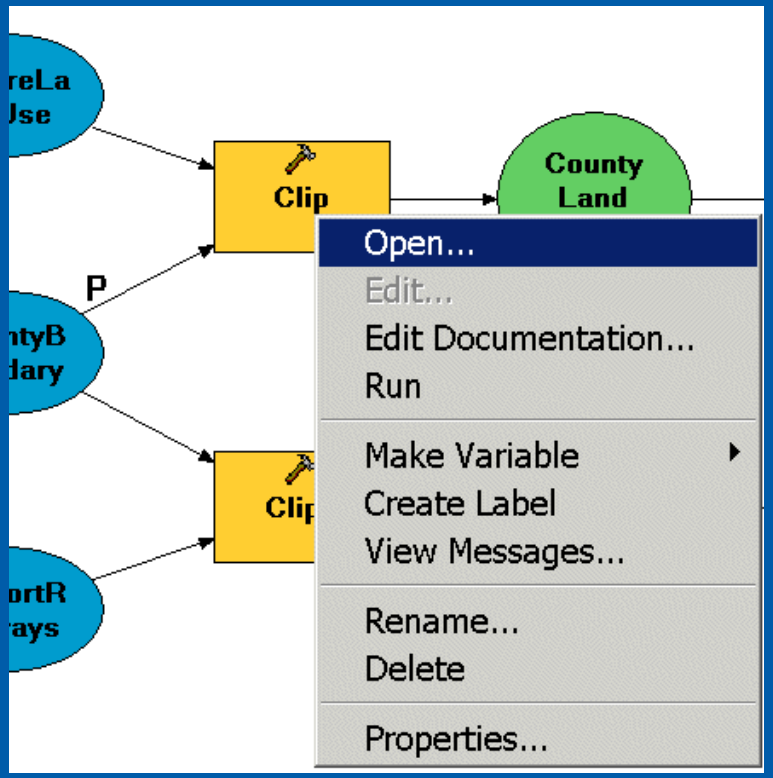
Adding model elements

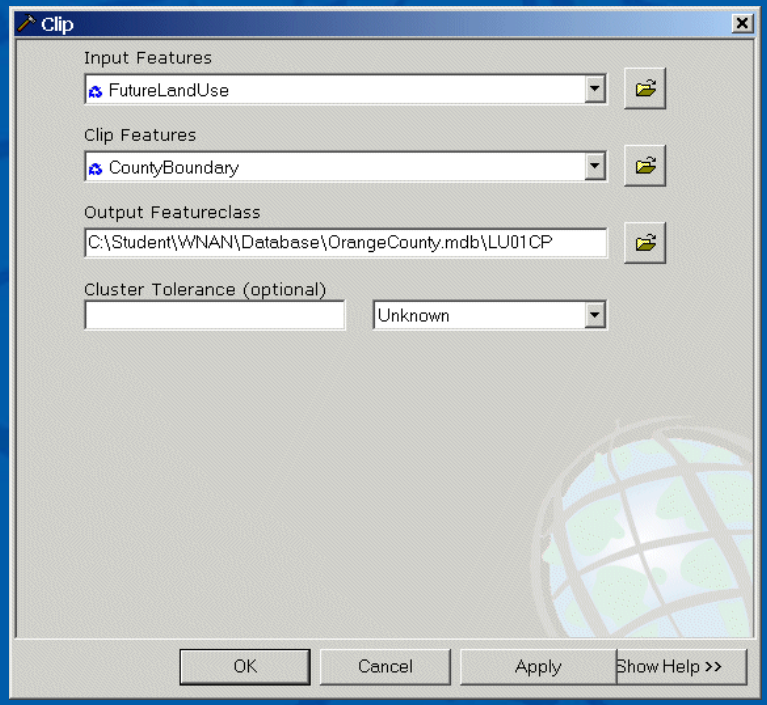
- Drag and drop from ArcCatalog or ArcMap
  - Tools from ArcToolbox
  - Data
- Add empty variables
  - Supply data source at a later time



Tools within a model

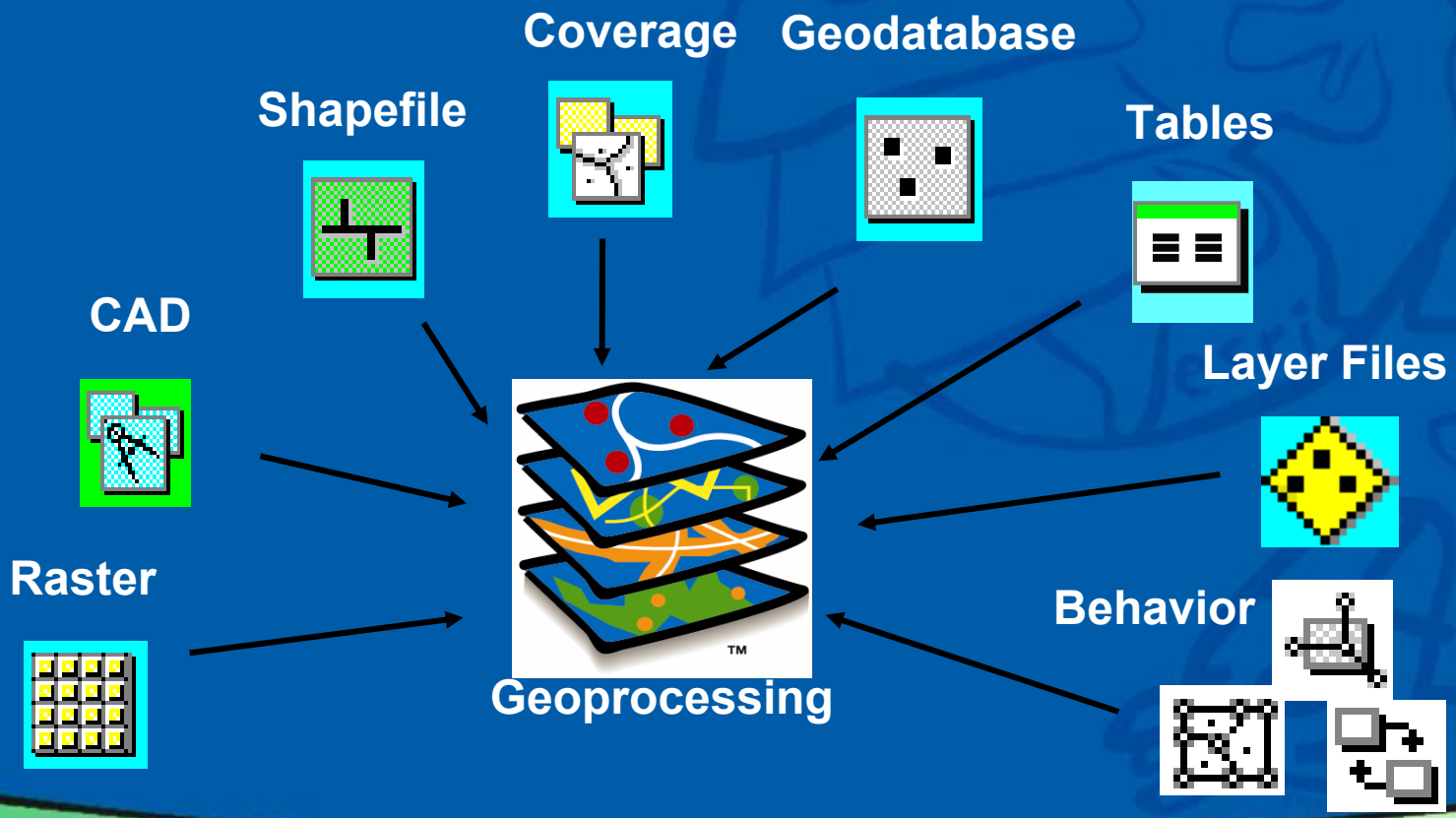
- Right-click or double-click to obtain parameters
  - Same dialog as tools from a toolbox





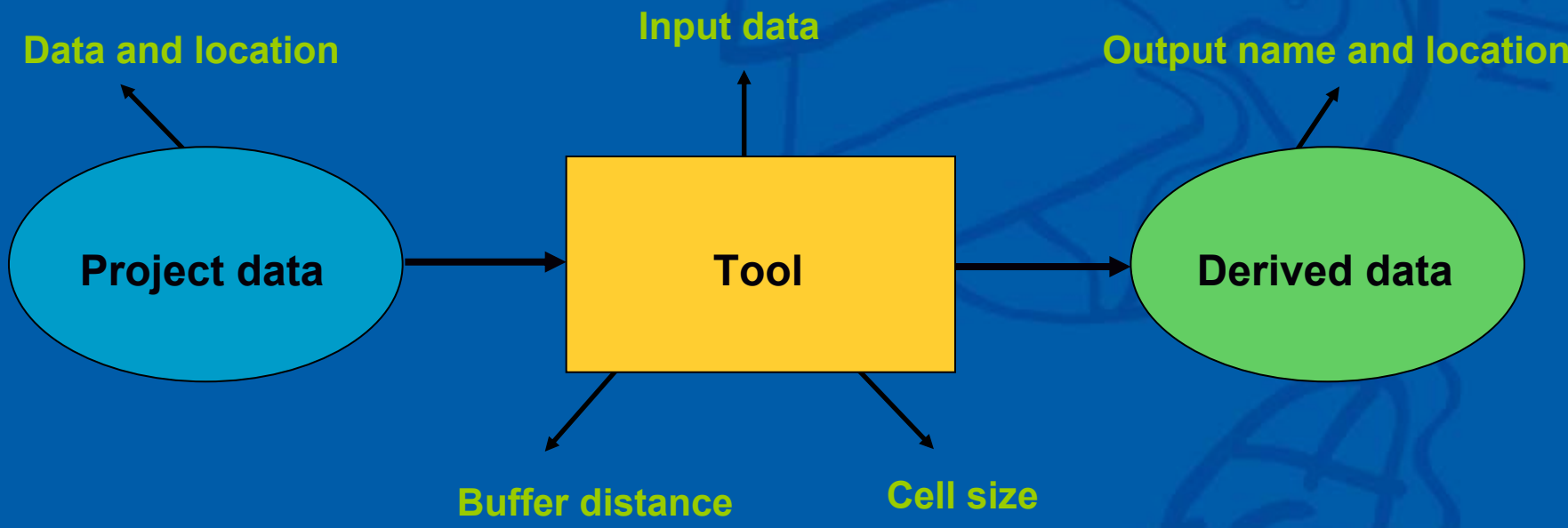
Supported data types

- Works with all data types used in ArcGIS
- Drag and drop data into model from ArcCatalog



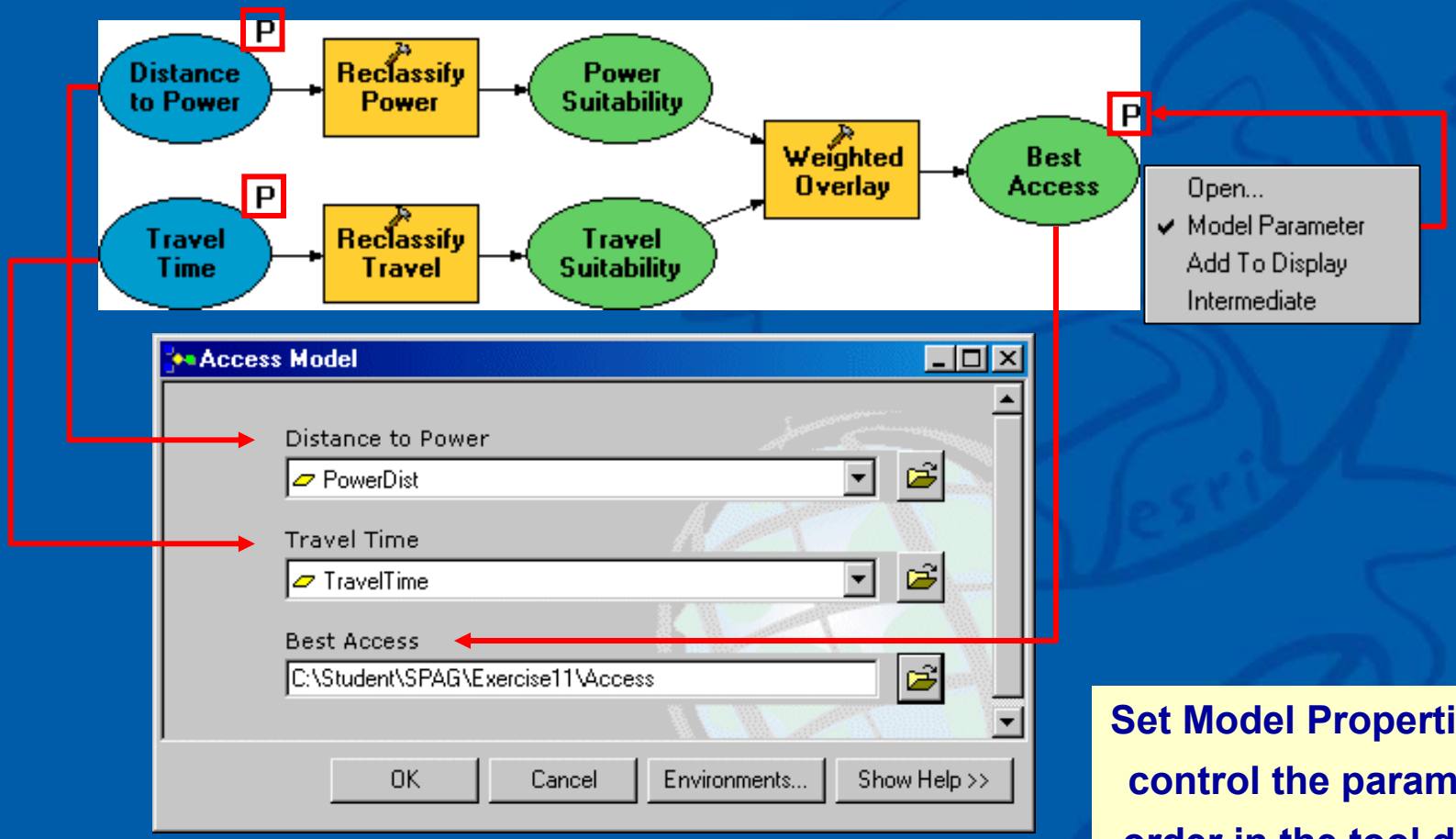
Parameters

- Input/Output data and values for a tool
- Used for running model as dialog
- Right-click model element and choose to create parameter



Running models with parameters

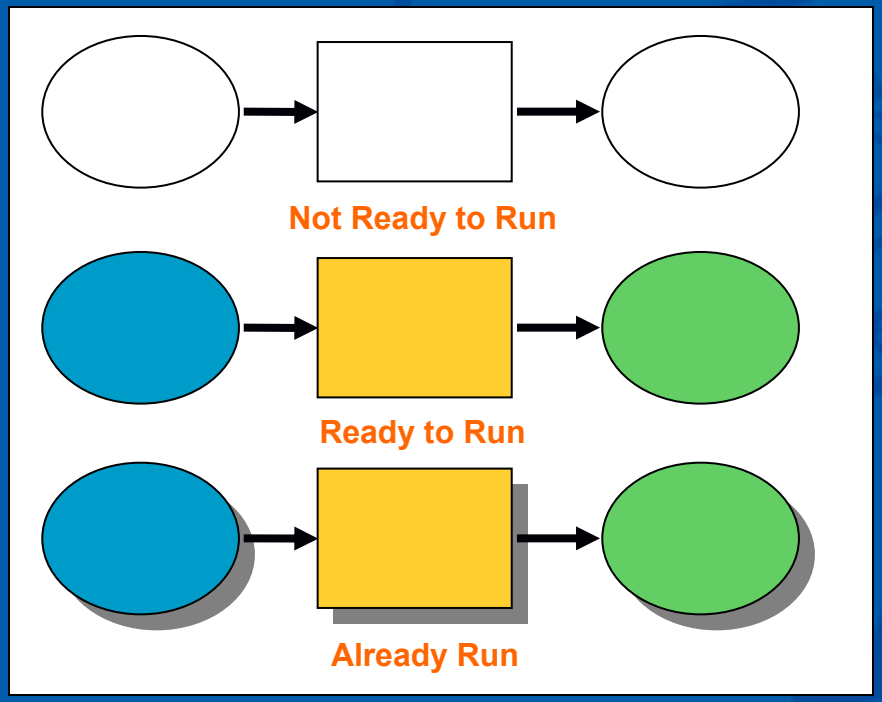
- All parameters in model appear in model dialog



Set Model Properties to control the parameter order in the tool dialog

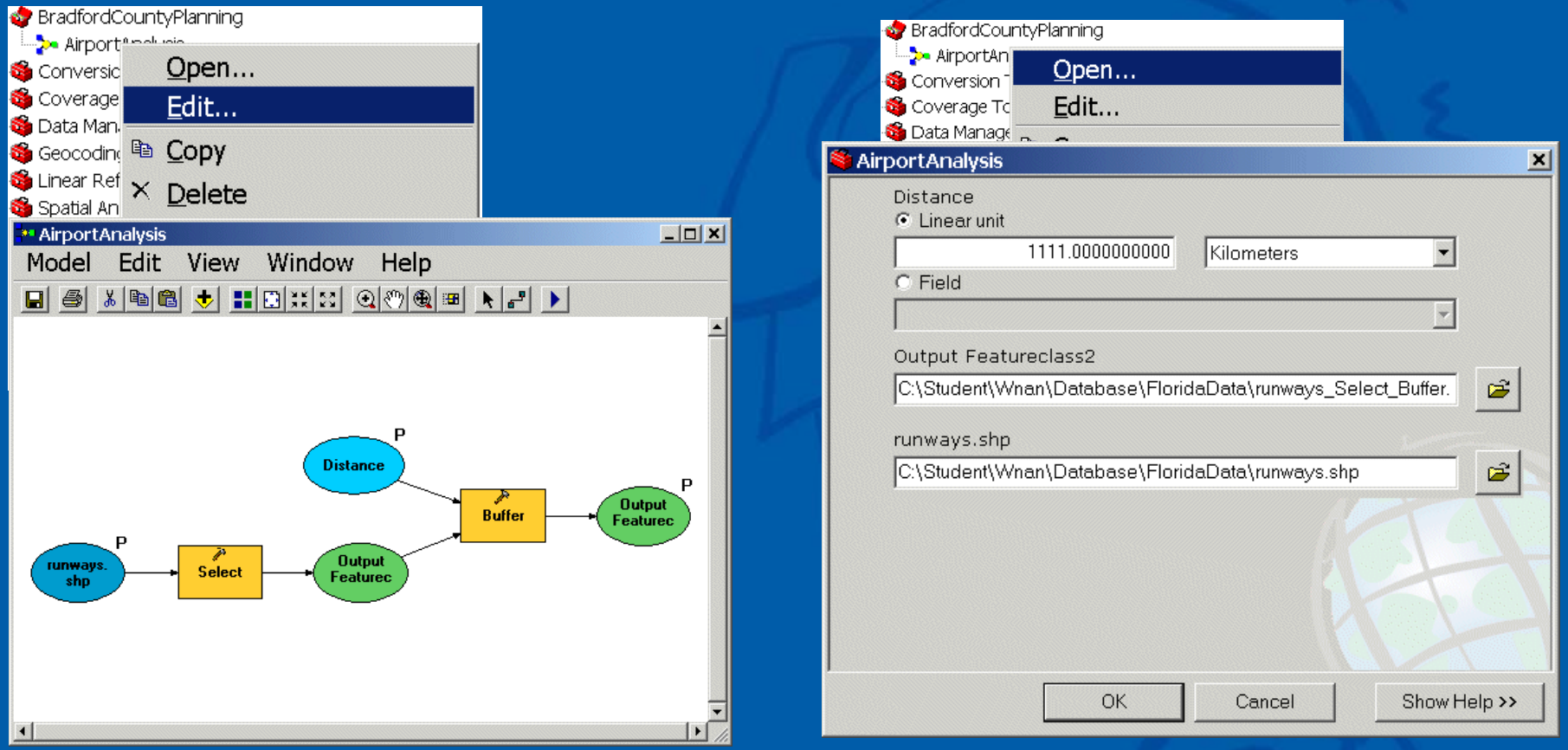
Three states of model elements

- Not ready to run: Parameters not defined
- Ready to run: All elements colored
- Already run: All elements colored and shadowed



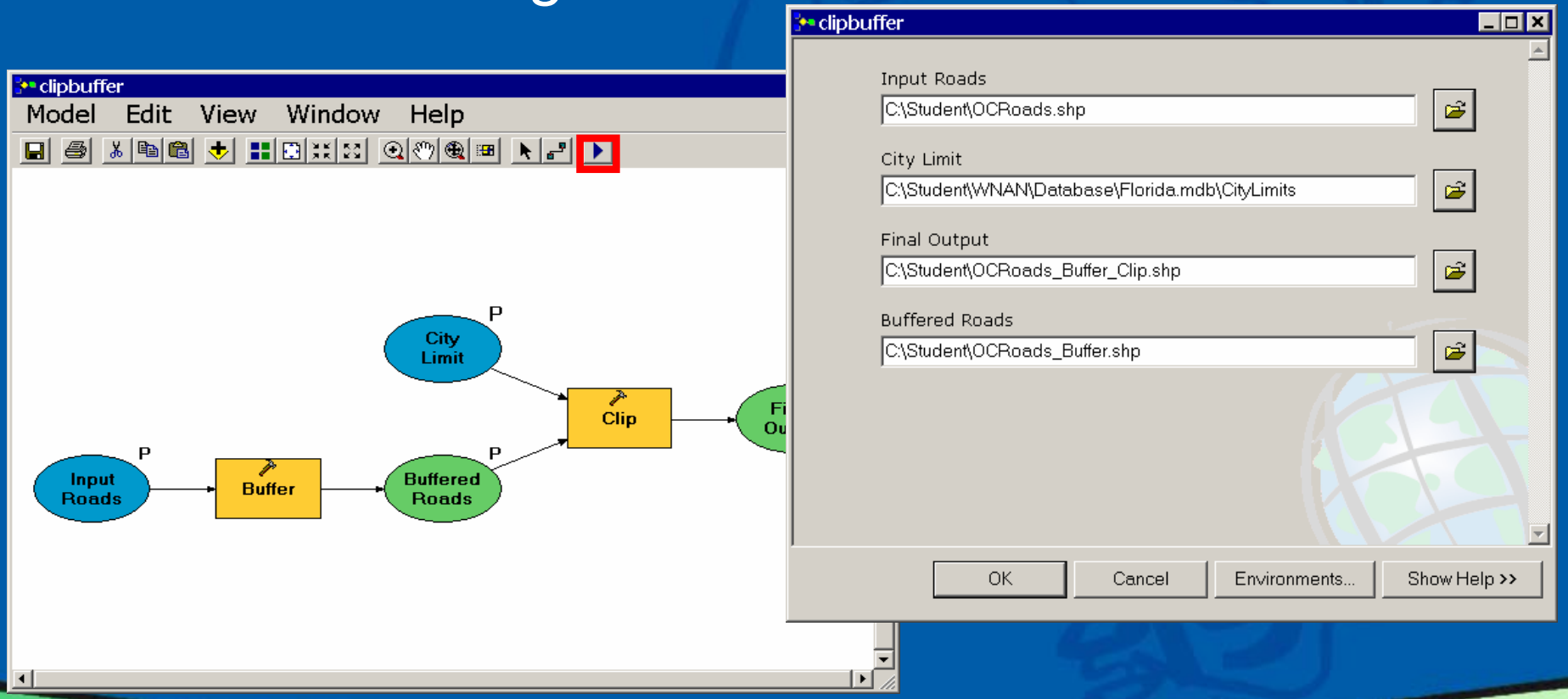
Running models

- All parameters created in model appear in model



How to execute models

- Entire model from ModelBuilder
  - Can execute one process at a time from ModelBuilder
- As a tool dialog

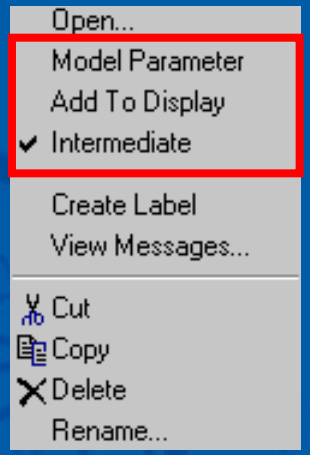


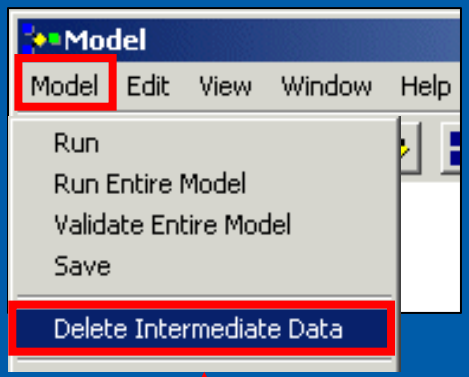
Demonstration 1

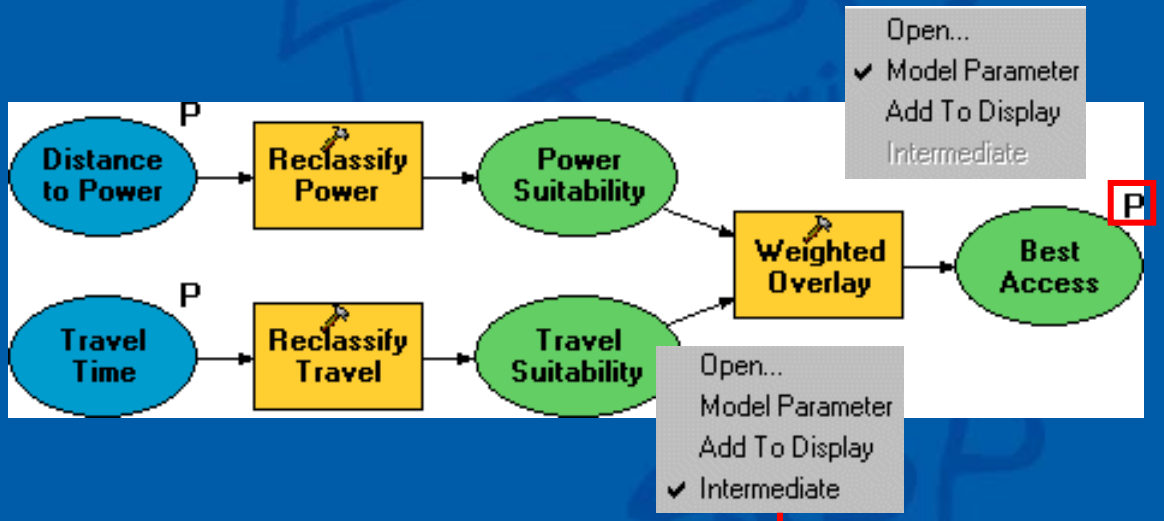
- Create a toolbox
- Create and run a simple model
  - Drag and drop tools and data
  - Parameters

Managing derived data

- Set as model parameter
- Add to display
  - Derived data is added to ArcMap display
- Intermediate (default)
  - Deleting Intermediate data deletes all output data flagged as intermediate

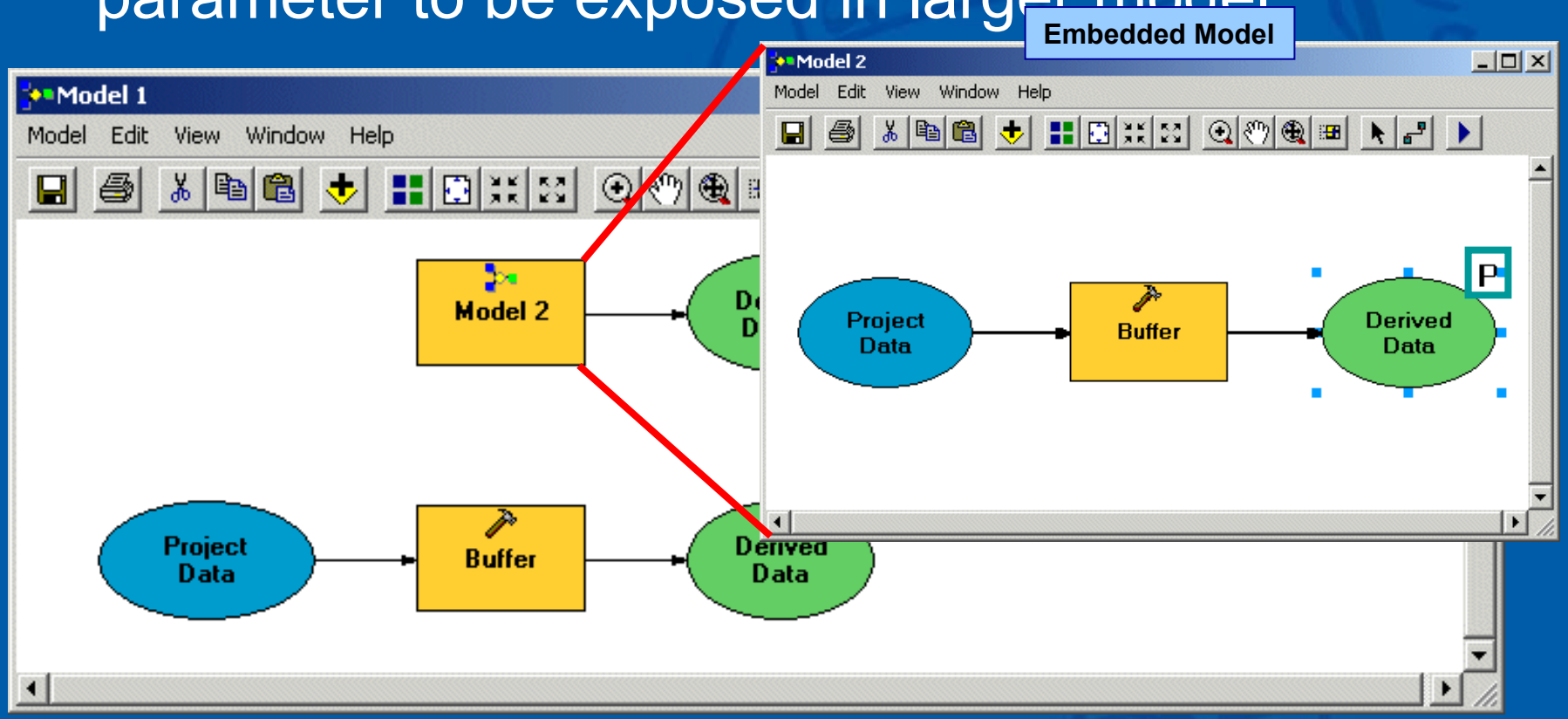






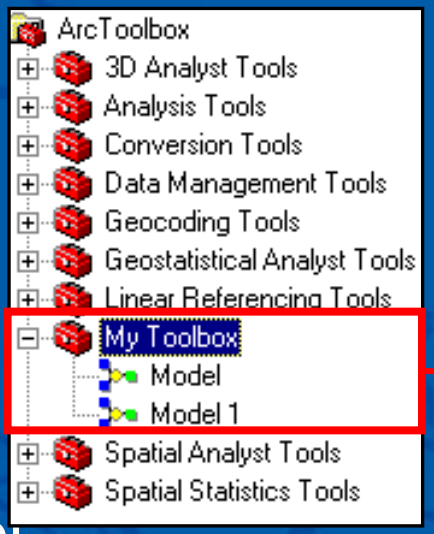
Embedding models

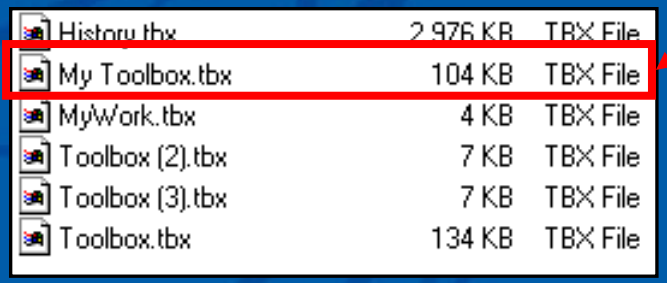
- Drag and drop model from toolbox
- Output from embedded model must be model parameter to be exposed in larger model



Saving and sharing models

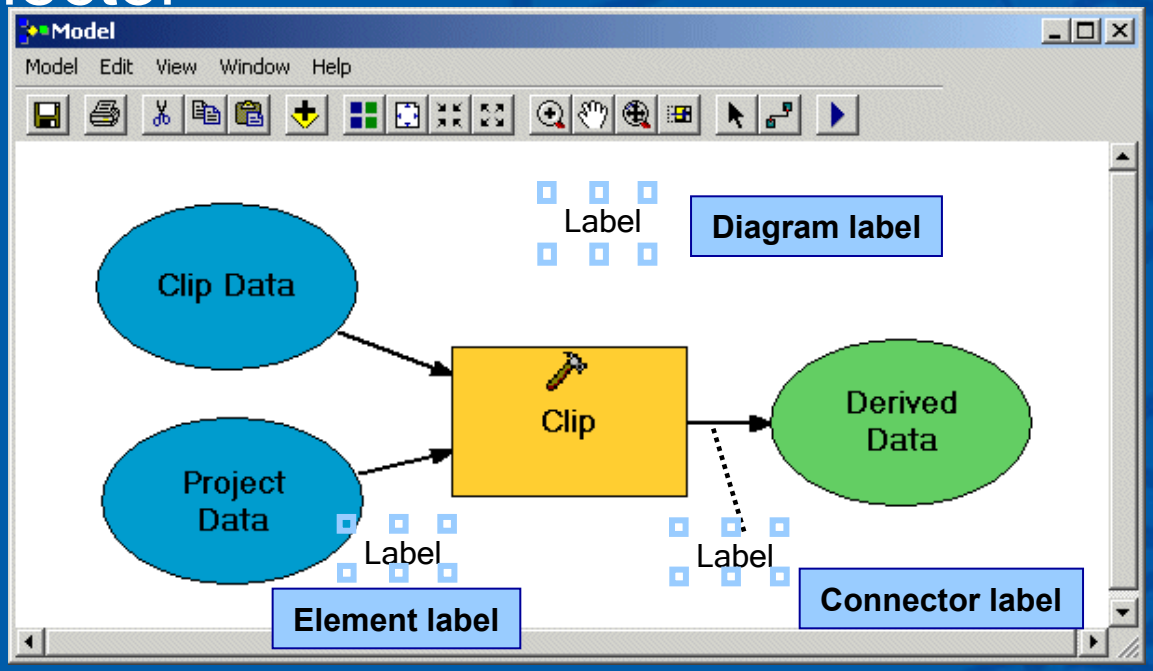
- Why share models
  - Collaboration
  - Refine and standardize models
- Model is saved to .tbx file or in geodatabase
  - Give .tbx or geodatabase to share model
- Set model parameters if used with different data





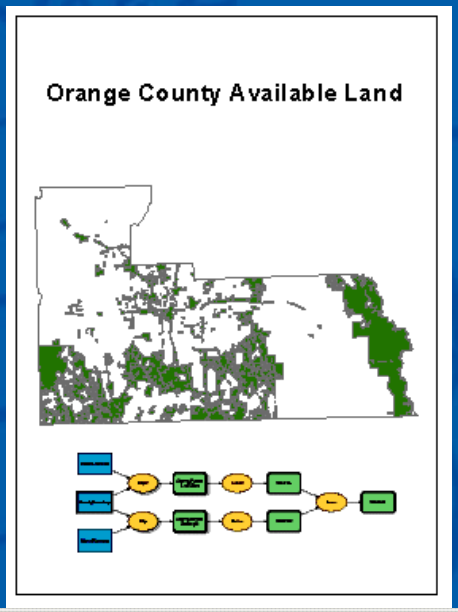
Model labels

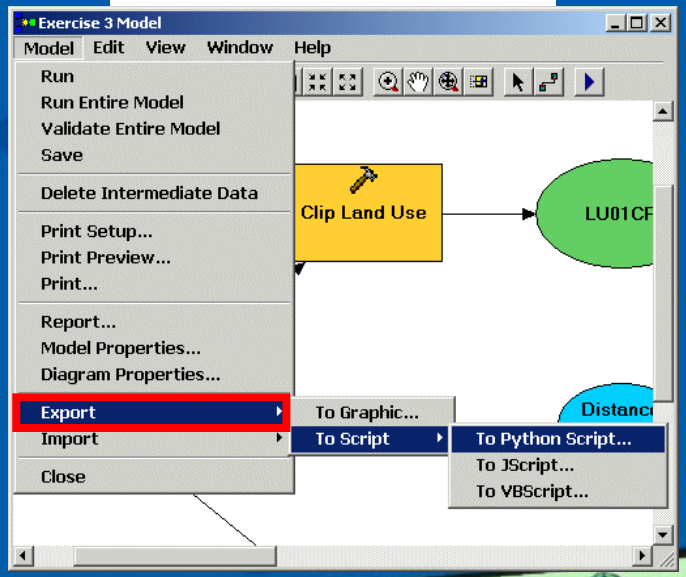
- Diagram: Free-floating labels
- Element: Maintain position relative to model elements
- Connector (tool or data): Maintain position relative to connector



Exporting and printing models

- Export to a graphic
  - Export as .bmp, .jpg, and .emf
  - Place in map layouts
- Export models to scripts
  - Python, JScript, and VBScript
- Printing models
  - Modify print settings Area to print
    - Number of pages on which the model will print





Demonstration 2

- Show how to share the model
- Export to python script

Introduction to Scripting and Python

Python

- Why use scripts and Python?
- Python code structure
- Selected geoprocessing tools
- Getting help with writing scripts
- Batch processing
- Making a script dynamic

Why write scripts for geoprocessing?

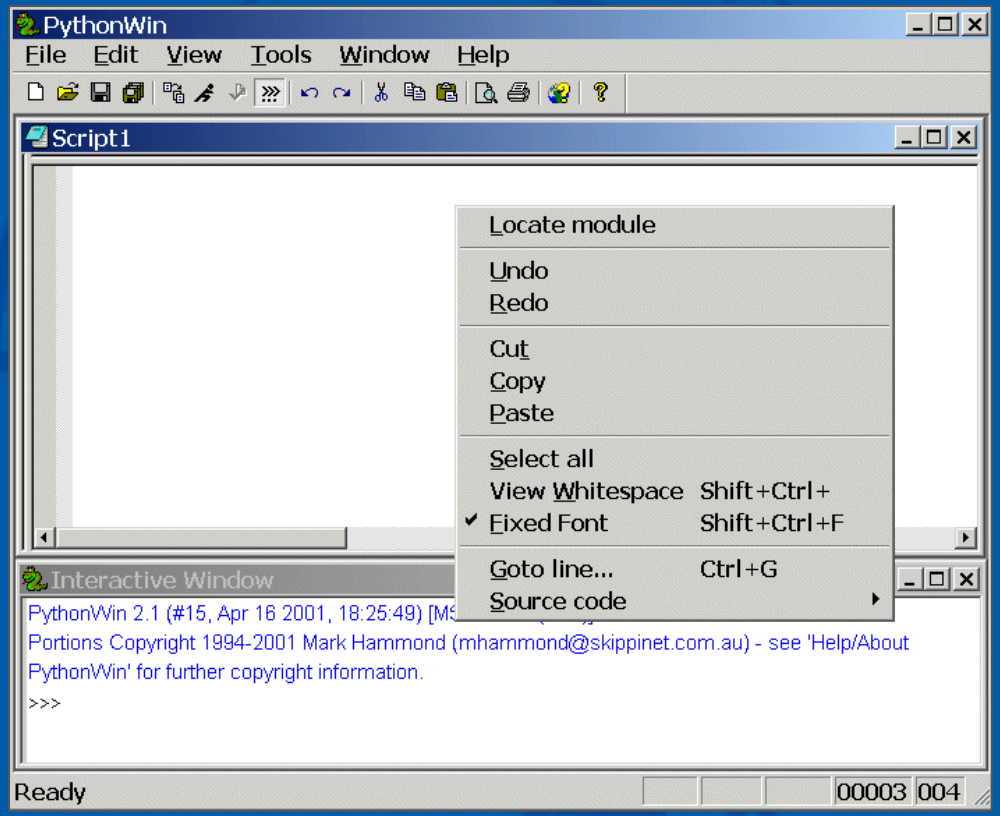
- Similar advantages that models have
  - Efficiently execute series of different tasks
  - Easy to read and document
  - Easy to share
- Perform batch operations
- Self contained (single file)
- Run any time
- Familiar environment for AML and Avenue users

Why use Python?

- Flexible, easy scripting language
- Object-oriented
- Offers a debugging environment
- Modular: Can be broken apart
- Cross platform
- It is FREE
- ESRI samples provided

PythonWin interface

- Menus, toolbars, and context menus
- Script window
  - Write and save code
- Interactive window
  - Test lines of code
  - Report messages
- Benefits
  - Windows look and feel
  - All in one application
  - Script tools open in PythonWin



Python Overview

- The geoprocessor
- Writing scripts
  - Comments
  - Variables
  - Syntax
  - Strings
  - Numbers

Writing scripts

- Import COM client support

```
import win32com.client
```

- Instantiate the Geoprocessor object

```
gp = win32com.client.Dispatch("esriGeoprocessing.GPDispatch.1")
```

- Set properties (e.g., workspace)

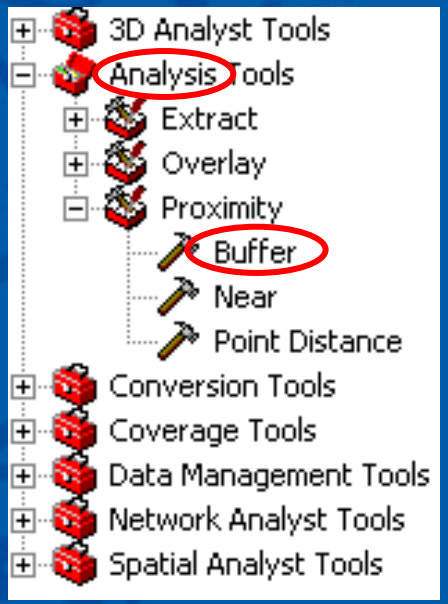
```
gp.workspace = "c:\\Florida.mdb"
```

- Comment code

```
# Buffer roads by 100 meters
```

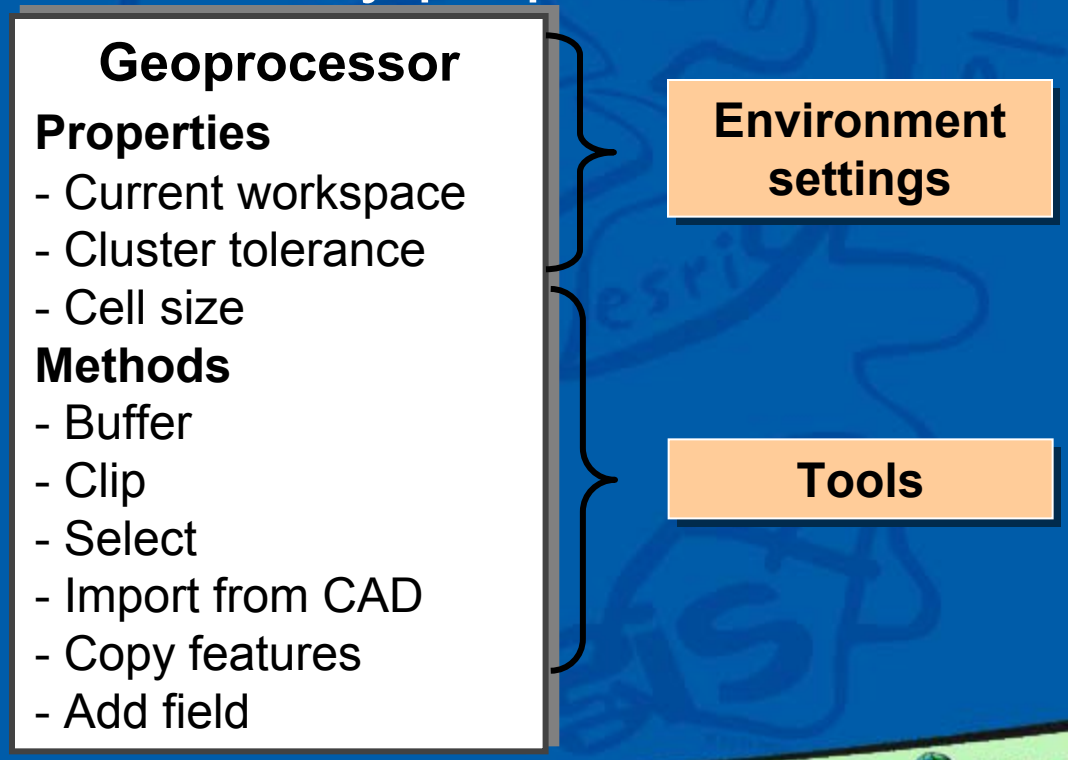
- Run tools

```
gp.Buffer_analysis("roads", "rdbuf100",
 "100")
```



The Geoprocessor ArcObject

- Most geoprocessing functionality on one ArcObject
  - Geoprocessor (GpDispatch)
- The Geoprocessor has many properties and methods



Syntax for properties and methods

- To assign a value to a property

  Object.Property = Value

  ```
  gp.Workspace = "C:\\temp"
  ```

- To get the value of a property

  Object.Property

  ```
  gp.Workspace
  ```

- To use a method

  Object.Method (arg, arg, …)

  ```
  gp.Buffer_analysis (fc,"C:\\temp\\buff.shp",
    100)
  ```

  – Parentheses around arguments

  – Arguments separated by commas

Comments

- Comment: A non-executable line of code
  - # sign
  - Comment and uncomment blocks of code to control execution

```
# Date: July 11, 2005
# Purpose: To buffer a feature class.
import win32com.client
gp =win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")
gp.Workspace =
  "C:\\Student\\PYTH\\Database\\SanDiego.mdb"
## Buffer the Freeways feature class 1000 feet.
gp.Buffer_analysis ("Freeways", "BuffFreeway", 1000)
```

Example: The Buffer tool

- Syntax

```
Buffer_analysis (in_features,
  out_feature_class, buffer_distance_or_field,
  line_side, line_end_type, dissolve_option,
  dissolve_field)
```

- Example

```
gp.Workspace = "D:\\AGIC2005"
gp.Buffer_analysis("Highway.shp",
  "BuffHighway.shp", "100 feet")
```

- Notes
  - If specifying units, make the argument a string
    - If units not specified, input feature class units used

Example script

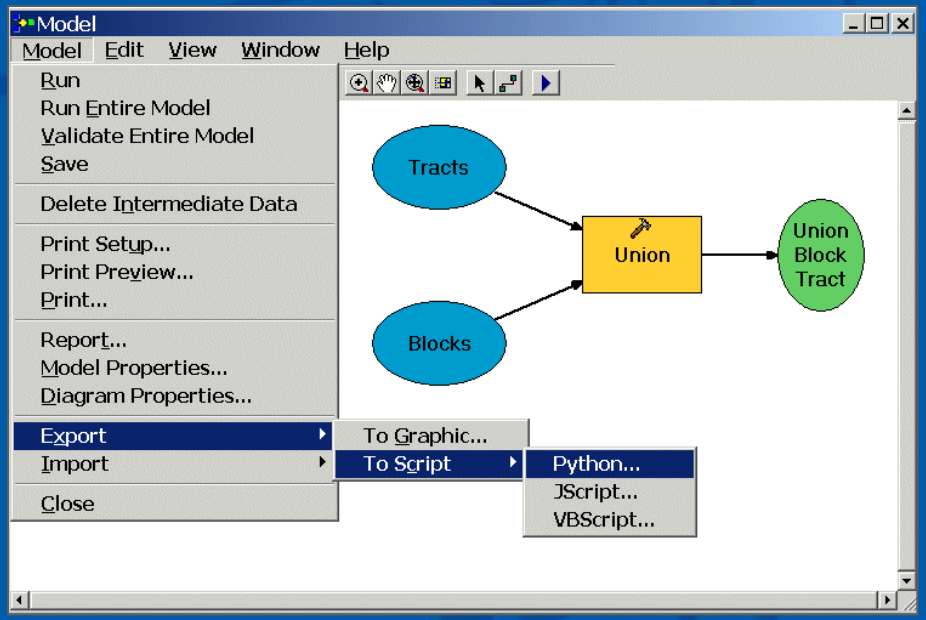
```
# Import COM support
import win32com.client
# Create the Geoprocessor object
gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")
roads = "C:/Data/Florida/ROADS.shp"
Out_Buff = "C:/Data/Florida.mdb/ROADS_Buffer"
citylimit = "C:/Data/Florida/citylimit.shp"
Out_Clip = "C:/Data/Florida.mdb/BUFF_Clip"

# Process: Buffer...
gp.Buffer_analysis(roads, Out_Buff, "10", "FULL", "ROUND")
# Process: Clip...
gp.Clip_analysis(Out_Buff, citylimit, Out_Clip)
```

Learning how to populate tool arguments

If I want to use the UNION tool, how would I know that the inputs are separated by semicolons?

1. ArcGIS Desktop Help
2. If help is not detailed enough, export the tool from a model to a script



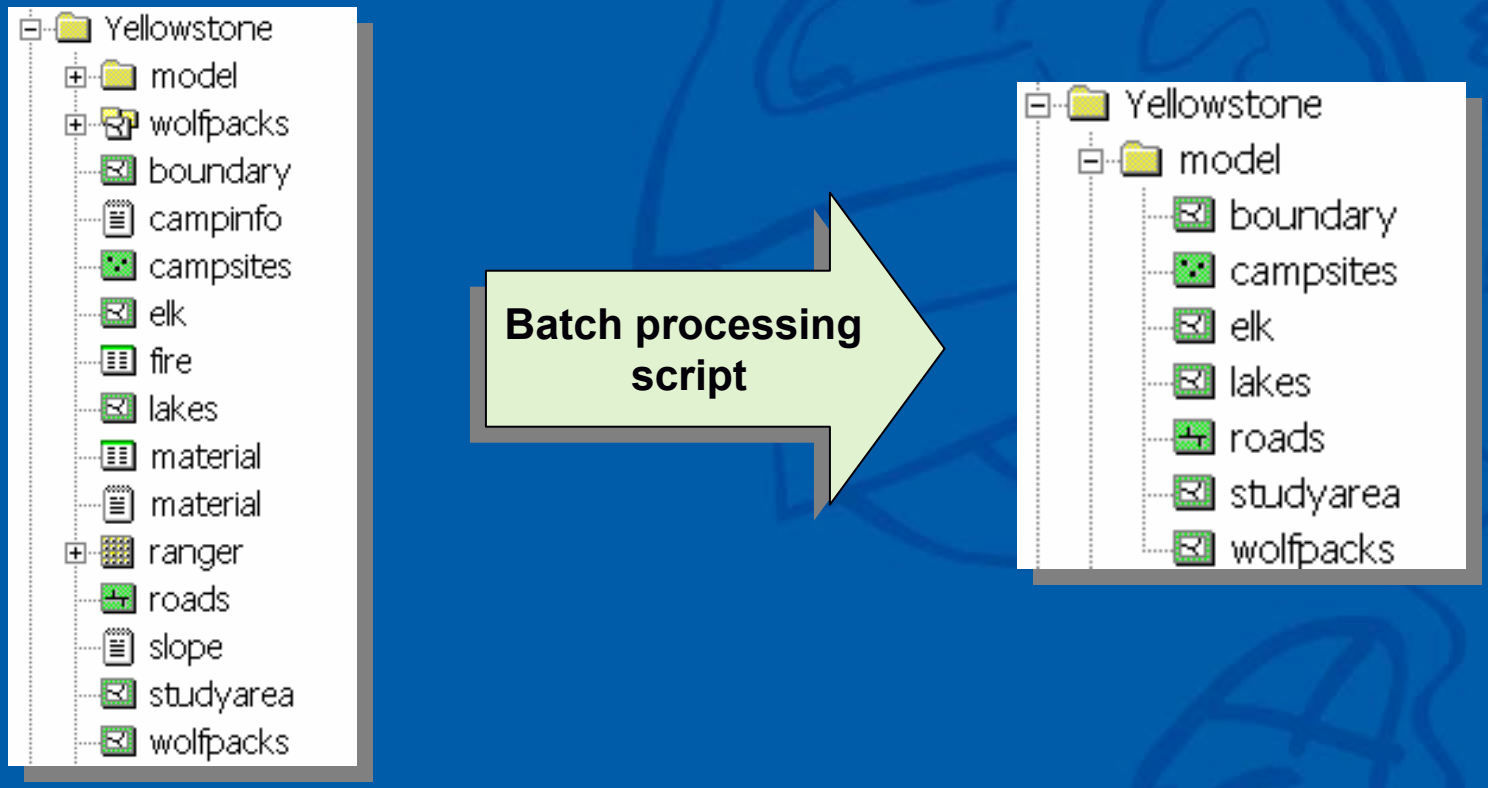
Demonstration 3

- Where to find help – ArcGIS Desktop Help
  - Geoprocessing tools
  - Geoprocessor object methods
- Running the a geoprocessing tool

Batch Processing
and advanced Scripting functionality

Batch processing

- Scripts are ideally suited for batch processing
- Example: Clip all shapefiles in a folder to a boundary



- Rerun script when new data is added to folder

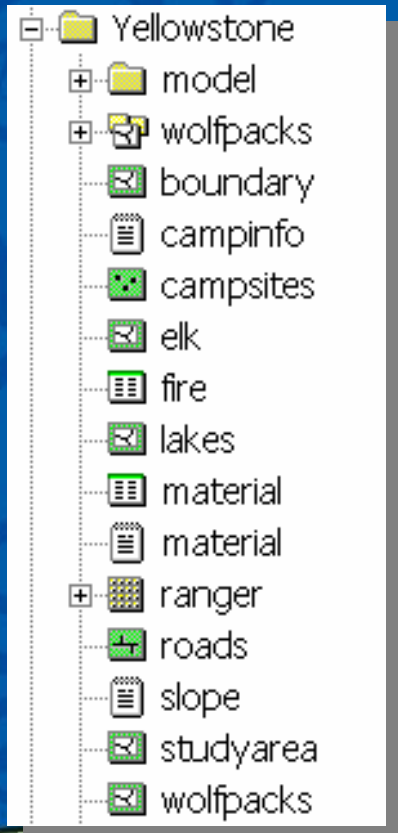
Listing data

- Enumeration: Lists of objects without a known count
- Use a looping structure to process one object at a time
  - ListFeatureClasses (Wildcard (optional), Type (optional))

```
#return a list of shapefiles in a workspace
gp.workspace = "C:\\Yellowstone"
fcs = gp.ListFeatureClasses("*", "all")
```

- Examples of Enumerations
  - ListFeatureClasses
  - ListFields
  - ListTables
  - List Indexes
  - ListRasters
  - ListWorkspaces
  - ListDatasets

fcs = all shapefiles in C:\Yellowstone

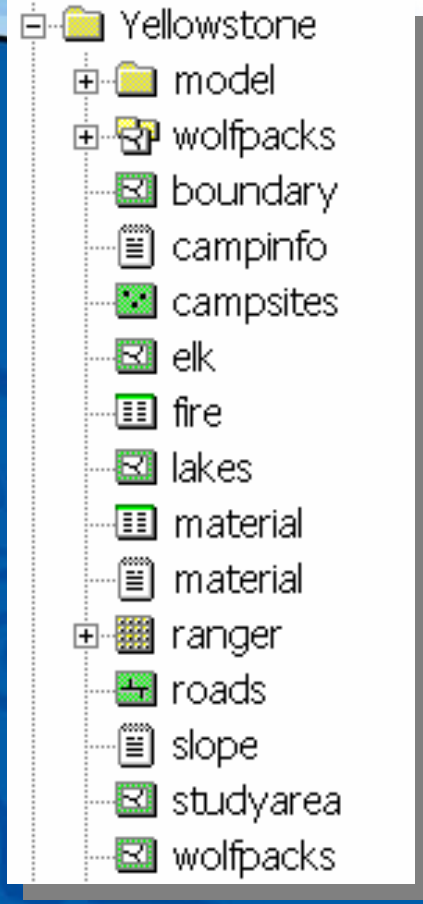


Looping

- While a condition is true

```
gp.workspace = "C:\\Yellowstone"
fcs = gp.ListFeatureClasses ("*", "all")
fc = fcs.next()
clipfc = "C:\\Yellowstone\\studyarea.shp"
outws = "C:\\Yellowstone\\model"
while fc != "":
    gp.clip_analysis(fc, clipfc, outws + "\\" + fc)
    fc = fcs.next()
```

- Loop is defined by indentation in Python
- Indentation is a language construct in Python
  - Needs to be consistent
  - Use <Tab> or spaces



Making Scripts Dynamic

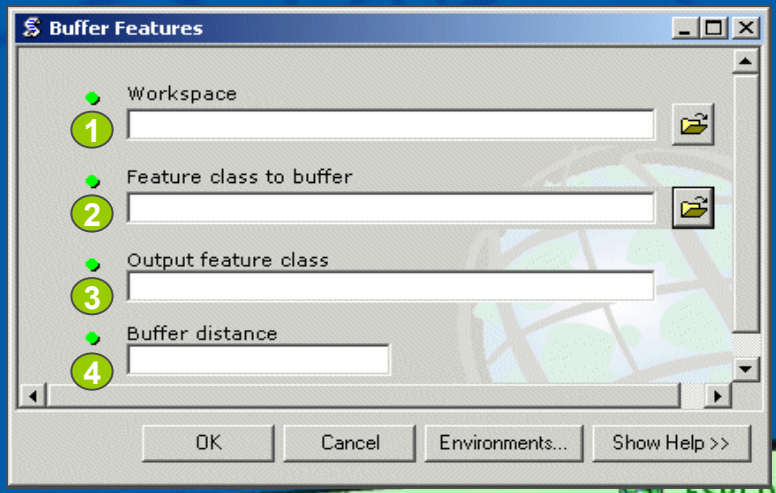
Making scripts dynamic

- Scripts can be static or dynamic
  - Up until now, all your scripts have been static
- Can add arguments to make a script dynamic
  - Let user run from ArcToolbox
- Two ways to create arguments
  - Python has a function called `sys.argv[]`
  - The Geoprocessor has a method called `GetParameterAsText()`

Creating arguments: `sys.argv[]`

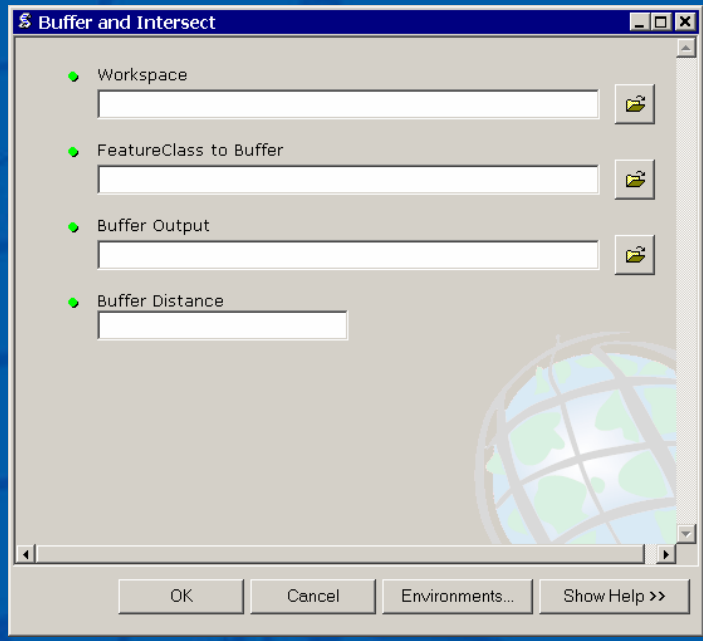
- Need to import the `sys` module
- First argument starts at `1`
- Run from ArcToolbox, PythonWin, or Command Prompt
- Limit to the number of characters

```
import win32com.client, sys
gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")
gp.Workspace = sys.argv[1]
bufFC = sys.argv[2]
bufOut = sys.argv[3]
bufDist = sys.argv[4]
```



Why use parameters with script tools?

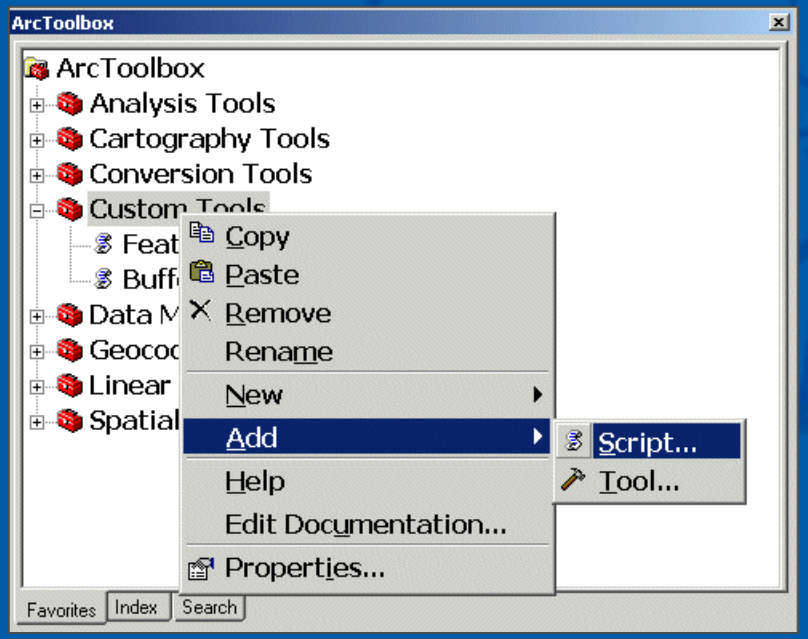
- Makes scripts more flexible
- Dynamically assigns script args
  - Workspace
  - Feature classes
  - String, numbers
- Write scripts to capture arguments



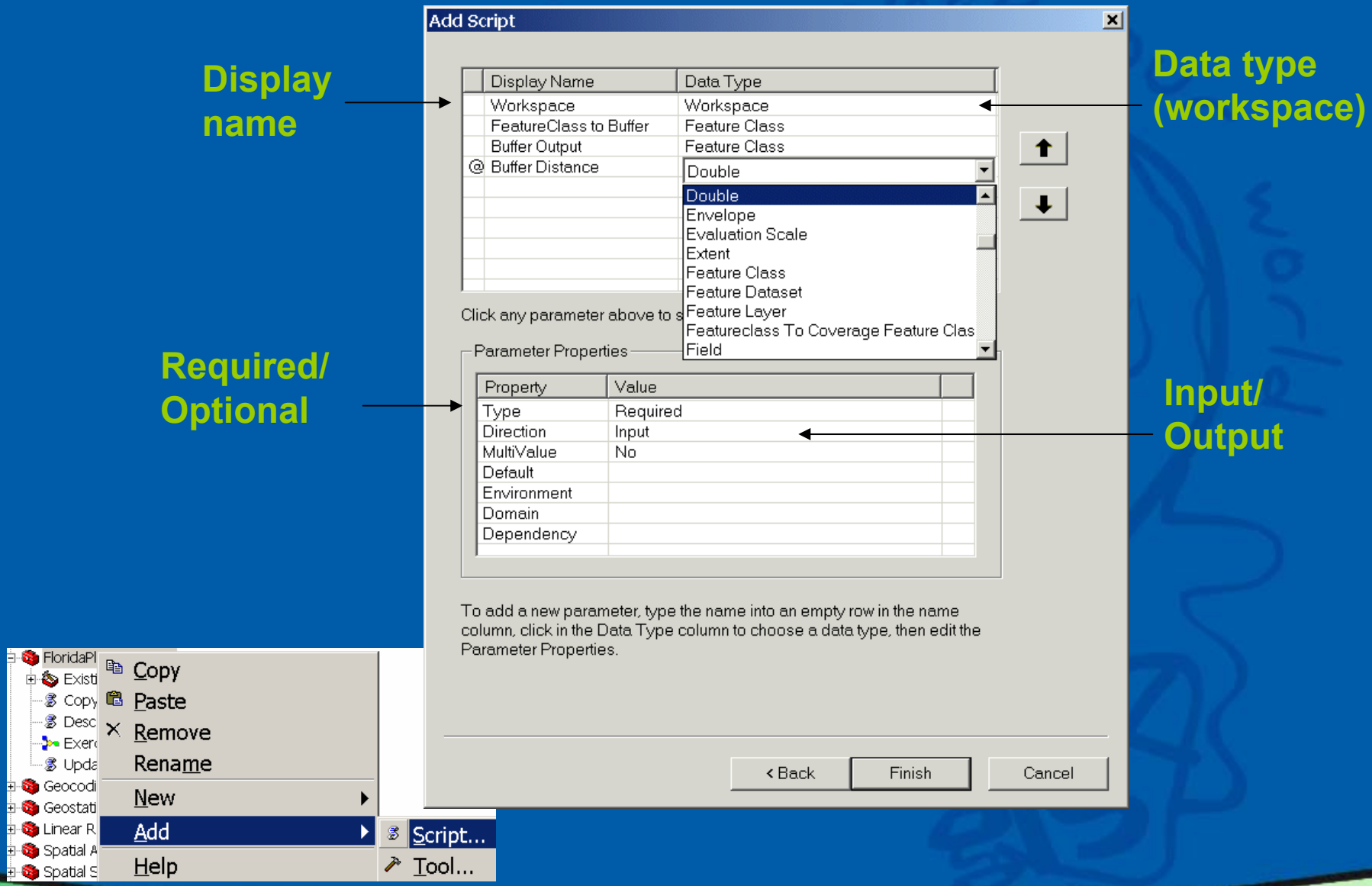

```
import win32com.client, sys
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")
gp.workspace = sys.argv[1] # first argument
buffc = sys.argv[2] # second argument
buffoutput = sys.argv[3]
buffdistance = sys.argv[4]
```

Attaching a script to a tool

- Use a custom toolbox
  - System toolboxes are Read-only
- Right-click the toolbox and click Add > Script



Parameter properties



Demonstration 4

- Add parameters to a script
  - Sys.argv[]

Resources for learning Python

- Books
  - *Learn to Program Using Python*
  - *Learning Python*
  - *The Quick Python Book*
  - *Python, Essential Reference*
- Web sites
  - The Python Foundation (www.python.org): Tutorials, documentation
- ESRI Instructor-led course
  - Introduction to Geoprocessing Scripts Using Python
- *Writing Geoprocessing Scripts with ArcGIS .pdf*
- Online help

Thanks!!